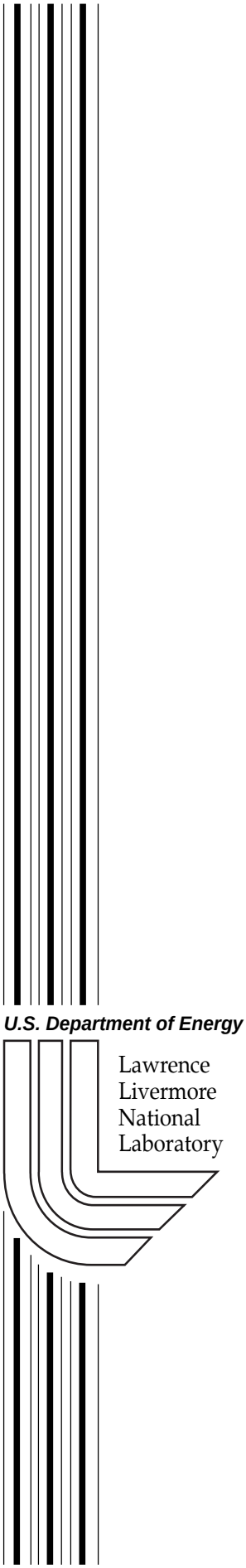
UCRL-JC-151932

A Multiresolution Image Cache for Volume Rendering

E. LaMar, V. Pascucci

This article was submitted to Eurographics 2003, Granada, Spain, September 01, 2003 – September 06, 2003



February 27, 2003

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

This research was supported under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

A Multiresolution Image Cache for Volume Rendering

27th February 2003

Abstract

We discuss the techniques and implementation details of our shared-memory image caching system for volume visualization and iso-surface rendering. One of the goals of our system is to decouple image generation from image display. This is done by maintaining a set of impostors for interactive display while the production of the impostor imagery is performed by a set of parallel, background processes.

Our system introduces a caching basis that is free of the gap/overlap artifacts of earlier caching techniques. Instead of placing impostors at fixed, predefined positions in world space, our technique is to adaptively place impostors relative to the camera viewpoint. The positions translate with the camera but stay aligned to the data; i.e., the positions translate, but do not rotate, with the camera. The viewing transformation is factored into a *translation* transformation and a *rotation* transformation. The impostor imagery is generated using just the translation transformation and visible impostors are displayed using just the rotation transformation.

Displayed image quality is improved by increasing the number of impostors and the frequency that impostors are re-rendering is improved by decreasing the number of impostors.

keywords: image cache, impostors, scientific visualization, multiresolution techniques, hierarchies techniques

1 Introduction

Scientists are faced with a problem that as their simulations grow to sizes where interesting features and structures can be resolved, the features themselves become to small (relative to the size of the data) to find, and the structures too large to visualize. Interactive navigation and exploration of these datasets is essential, and showing both large scale structures and small scale features in the same visualization is essential. However, the data size prevents efficient rendering of even single frames, let alone multiple frame per a second that is required for interactive exploration. While multiresolution techniques applied to the source data can improve rendering speed, it still requires that the entire approximation be rendered every frame. For some newer datasets, even minimalistic approximations are too large to be rendered interactively.

Caching and resusing imagery over several frames to amortize the cost of rendering that imagery is proving to be a very usefull technique (see section 2).

We have developed a system to cache imagery for volume visualization and iso-surface rendering that addresses issues specific to the use of impostors scientific visualization (section 3).

Our system decouples generation of imagery from the display of the imagery, and decouples the resolution of generated imagery from the resolution of displayed imagery . Impostor is the name for the entity that associate a cached image with a spatial position and extent (and temporal position and extent for time varying data sets). Secondly, our system also decouples the placement of impostors from the specifics of the data layout; that is, the impostors placed to reflect user interest and rendering concerns. Our system introduces a caching basis that is free of the *gap* and *overlap* artifacts of billboard techniques (section 4). Instead of placing impostors at predefined positions in world space, our technique is to place impostors at positions relative to the camera viewpoint, that move with the camera viewpoint. We call this a *cube-cache*.

Our system's structure 5 uses shared memory

symmetric multiprocessor machine to communication messages and images, a display process to render impostors, and multiple generator processes to produce impostor imagery.

The results (section 6) are promising, with linear scaling for a demonstration software based generator process. Limitations of hardware and software prevent use from studying scaling for real scientific data. There are a number of items (section 7) we need to add to our system to make it a viable tool for scientists.

2 Related Work

Bethel et.al. [Bet00, BSL+00] discuss the Visipult system - a distributed, multiresolution visualization system for time-varying datasets. Imagery is produced in distributed set of computers. Visapult's rendering pipeline starts with a multiresresolution dataset, a bricked, cartesian grid. Imagery is produced and the brick-level and transmitting to the viewing system. While they do not mention it, their system must have the *gap* and *overlap* artifacts because the placement of impostors is not held fixed, relative to the camera. The results for the work are very specific to the decomposition of the datasets and image generation is not independent of the data.

There is significant amount of work [CTFB01, CSKK99, SS96, SLS+96] on use and pre-processing of impostors for viewing geometric datasets. Significant preprocessing efforts to place impostors and determine visibility. Most typical applications are architecture walk-throughs, either of individual buildings, but more recently, of whole cities or city districts. These datasets are static are intended to be visualized many times, so it is reasonable to spend a large amount of preprocessing time to accelerate the rendering of them. Also, density and spatial extent of geometric datasets are rarely uniform, resulting is non-uniform placement of impostors. Often, significant user input is required to specify good impostor locations.

3 The Problem

Scientific data, however, differs is several significant ways, and techniques for caching imagery for geometric datasets do not work on them. A significant number of scientific datasets are uniform, cartesian grids, where the information density is very high and uniform. Thus, impostors density must be corresponding very high and uniform.

Densely placed impostors exhibit two related artifacts. The first is the *gap* artifact, and occurs when a large structure is represented by a set of adjacent impostors, then is viewed from a different position from where the impostors where originally generated. The impostors move apart from each other and no longer completely cover the large structure. When this occurs, regions hidden by this larger structure become exposed. The *overlap* artifact occurs when the impostors move together. Here, parts of the structure can cover, or overlap, itself.

It is not generally possible to determine good positions, *a priori*, especially when run-time, user-controlled parameters, such as transfer functions or isocontour values may significantly alter the appearance of the data. Thus, it is not possible to preprocess any imagery.

4 Caching Basis

Our system introduces a caching basis that is free of the *gap* and *overlap* artifacts of billboard techniques. Instead of placing impostors at predefined positions in world space, our technique is to place impostors at positions relative to the camera viewpoint, that move with the camera viewpoint.

We call this a *cube-cache*. Figure 1 shows an example *cube-cache* of a 2d cache viewed from above. The red dot at the center is the camera center; black lines show the base decomposition of the space around the camera; blue lines show the KD-tree decomposition of each quadrant; and green lines show individual impostors. The impostors are shown slightly smaller that their real physical extent (delimited by blue & black lines), as to emphasize that they are independent entities. Note that the different quadrants have different degrees of decomposition. Each face of the *cube-cache* is an independent KD-tree.

Note that the KD-tree decomposition planes are either parallel to the cube face or pass though the origin of the cube-cache. This is the reason that there are no gap or overlap artifacts: impostors run to the boundaries of a KD node are their end-points do not

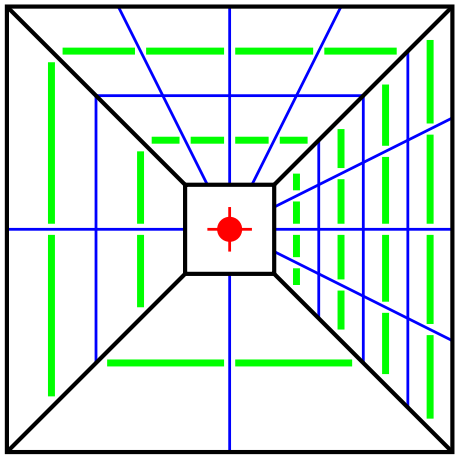


Figure 1: the cube-cache: 2d cache viewed from above. The red dot at the center is the camera center; black lines show the base decomposition of the space around the camera; blue lines show the KD-tree decomposition of each quadrant; and green lines show individual impostors.

move with respect to the camera.

Figure 2 shows where impostors would be reused between frames. In the first (left) frame, a set of impostors are rendered. The blue and green impostors are generated for (or before) the first frame. In the second frame, as the camera turns clockwise, another set of impostors are rendered. Those shown in green are rendered in both frames (ie, generated in the first frame and just reused in the second). Impostors shown in blue are not visible in the second frame, and may be deleted (if running out of cache space). The purple impostors in the second frame are now visible, so they may be placed in a *work* queue to be generated if their imagery is invalid.

4.1 Caching basis artifacts

There are two artifacts from this design. The first is that thethe projected size of the pixels are smaller as one approaches the edge and vertexes of the base cube. The second is when using nearest-neighbor filtering for low-resolution images, one can see the individual pixels and can see that they are not parallel. Figure 3 shows an example of both issues. Neither of these problems should really affect user perception of

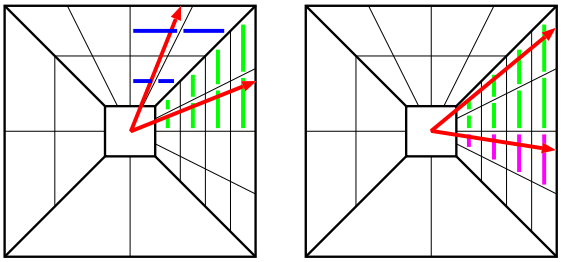


Figure 2: Reuse between frames: blue and green impostors are rendered in first frame. The green are reused in the second, with the purple newly visible. If the purple impostors are invalid, they will be queued for re-rendering.

a data space.

4.2 Rendering

The viewing transformation is factored in the a *translation* and *rotation* transformation. The rendering cycle consists of rendering and caching invalid, visible impostors just using the *translation* transformation, then rendering the impostors with just the *rotation* transformation.

All impostors have the same fixed resolution. An impostor is refined by replacing it with a left/right, top/bottom, or front/back impostor children, where each child contains a copy of the corresponding region of the parent impostor. The new impostors are marked for future re-rendered. Impostors are coarsened by removing the children and replacing the parent's imagery by a filtered version of the children's imagery.

Rotating the camera direction does not invalidate any of the impostors, but newly exposed impostors may require rendering, and possibly refinement. Impostors now partially exposed by to be coarsened. Our current system does not use any error metric for prioritizing impostors for re-rendering and visible impostors are re-rendered in a round-robin scheme.

5 System Architecture

Our system uses shared memory on a symmetric multiprocessor machine, a display process, and multiple

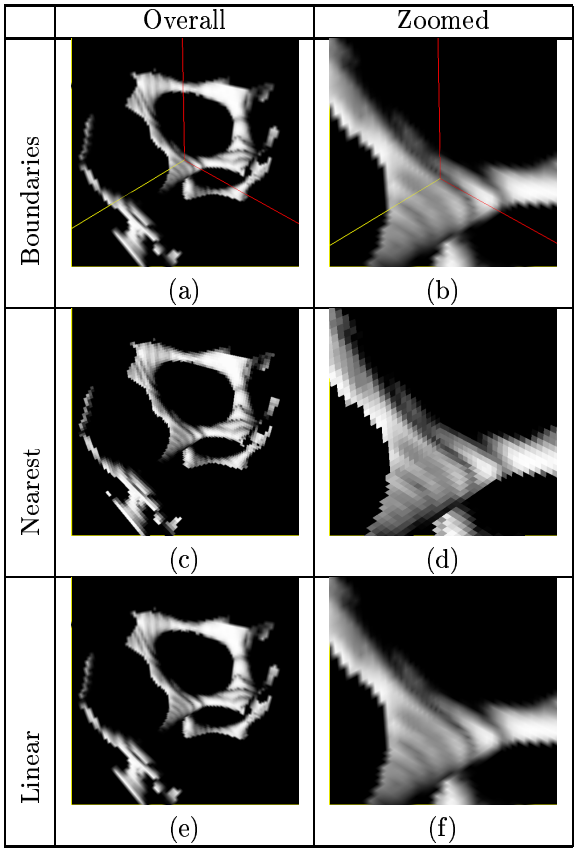


Figure 3: Artifacts of the *cube-cache* on a 64^3 voxel trebecular bone dataset. The first artifact is that the projected size of the texels are smaller at the edges and vertices of the base cube-cache. The second artifact is when using nearest-neighbor filtering for low-resolution images, one can see the individual pixels and can see that they are not parallel. Neither of these problems should really affect user perception of a data space. The yellow and red lines in figures (a) and (b) show the boundaries of impostors at one of the vertecies of the *cube-cache*. Figures (b), (d), and (f) show the magnified centers of figures (a), (c), and (e), respectively. Figure (d) shows the differing orientations of the textels using nearest-neighbor filtering. Figure (f) shows the affects of using linear filtering; the differing orientations are still visible, but much less obvious, and "limited" to the profiles.

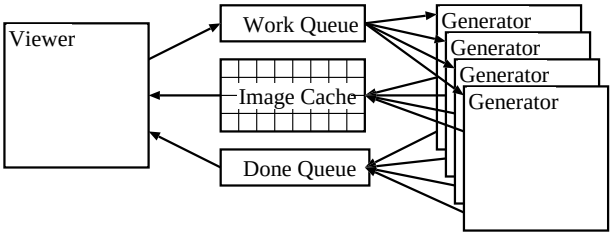


Figure 4: The system architecture. There is one Viewer processes and multiple Generator processes. The viewer writes requests to, and generators read requests from, the *work* queue. Requests are written back to the *done* queue. Generators write images to the image cache.

generator processes. We use process-level parallelism, rather than thread-level, due to issues with OpenGL thread safety and performance. Our system currently uses VTK as the rendering engine. Shared memory is implemented using the Unix *mmap* function call. The restriction is that the mapped file, and thus the image database, cannot be larger than 1GB; we have not found this to be an issue. Semaphores are the synchronization primitive. There can be one view program and multiple generator programs.

At start-up, a file is created that is large enough to store the image cache and the *work* and *done* queues.

The impostor database is just read by the display process and written by the generator processes. The *work* queue is written by the display process to request that impostors be subdivided, rendered, or merged. Generator processes only read from the *work* queue. The *done* is just written by the generator processes, and only read by the display process.

Both queues are fixed-size circular queues with the same number of elements. The available space in the queues is one less than the actual space allocated, as we use just address arithmetic (ie, comparing *head* and *tail* pointers), to determine if the queue is *empty* or *full*. We found that when we used a *size* field to calculate empty or full states, and running many generator processes, processes spent significant time blocking for the semaphore for the *size* field.

5.1 Generator processes

The generator processes obtain tasks (regions to render) from the *work* queue, render the region, place the image into the corresponding place in the image database, and write a write the task to the *done* queue. At this time, each generator process requires a display window be instanced on a desktop, somewhere. Offscreen rendering does not generally seem to work, and, according to the VTK documentation, rendering is performed entirely in software in offscreen mode. Several rendering modes are available: we use VTK's texture-based volume visualization, geometry-based iso-surface rendering, ray-cast-based volume visualization, and ray-cast-based iso-surface rendering.

5.2 Viewer process

The viewer program just renders impostors that are both within budget and are available. If an tile has no children, but does not meet the rendering requirements, it is added to the generator queue. The viewer process also reads the 'completed' queue to see what regions have been completed by the generator processes. When a region is completed, the viewer notes this in the image database.

We use this break-down of work to minimize the amount and granularity of sharing (via semaphores) that the processes need to perform.

5.3 Shared Memory Queues and Issues

We started with a very basic circular queue implementation for our queues, using two pointers to record the head and tail positions, and a size field to record the available space. Processes (both viewer and generator) must block to add or remove elements from the queue. The *work* and *done* queues are implemented as fixed-size circular queues. The queues are fixed size as it is not very easy to allocate more memory in a mmaped-shared memory scheme. A *size* attribute tracks the available space in the queues. Head and tail attributes track the location of the queue's head and tail; elements are added at the head and are removed at the tail.

We found that this basic implementation scaled very poorly, and when using eight (8) generator processes (and 1 viewer), the processes would occasionally (~1% of the time) block for as long as 0.5 second on a semaphore. This caused unacceptable stalls in the viewer process.

When using queues, where one process only writes to the queue and the other process only reads from the queue, means that the reader of a queue never modifies the head of the queue, and the writer of the queue never modifies the tail of the queue. Thus, the viewer and generators are now modify different attributes, relaxing the consistency constraint. That is, a reader process can only read from from a queue or test if it is empty; a writer process can only write to a queue or test if it is full. The test for a queue being full or empty is now conservative - a reader may find the queue empty, when, it fact, there may be something in it; similarly, a writer may find the queue full, when in fact there are some slots available.

The caveat is that reading and writing pointers on shared memory system must be atomic, and no partial values are returned. Thus, assuming that accessing a pointer is atomic, we can use pointer arithmetic to estimate queue size, and thus the viewer and generator processes never need to block on a shared semaphore. However, multiple generator processes must still use a semaphore to remove work from the *work* queue.

The "relaxed" model of a shared queue works correctly on the two systems that we have tested this system on, using 32-bit pointers on two processor Linux boxes and 48-processor SGI Origin3000.

5.4 Tasks and Messages

Three kinds of messages can be written to the *work* queue: subdivide, generate, and merge. When a generator process starts to service a message, it removes the message from the *work* queue. When a task is completed, the message is simply move to the *done* queue.

Subdivide

Subdivide an impostor and replace it with two children impostors that cover the same extent. An impostor can be split (with respect to the view direction) into left/right, top/below (see figure 5), or front/behind impostors pairs (see figure 5) . Since all impostors have the same resolution; when a left/right

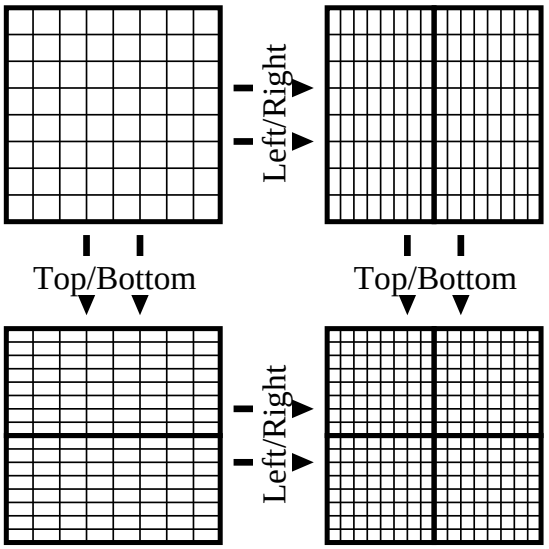


Figure 5: Refinement of a single impostors to pairs of children impostors. For example, a single impostors may first be split left/right, then top/bottom, or top/bottom, then left/right. Note that the texels in an impostor may have aspect-ratios other than one-to-one.

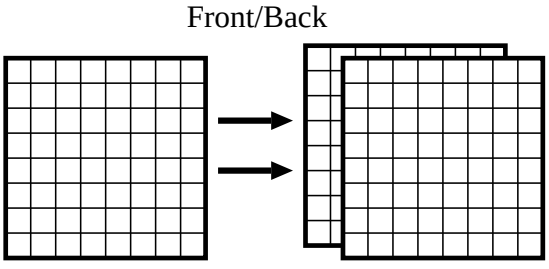


Figure 6: Front/Back split

or top/bottom pair is generated, the effect is to double to number of texels in the direction of the split.

For a front/back split, we factor the impostor into a front/back pair by reversing the compositing OVER operation. The OVER operator is defined as follows:

$$K = C_0 \, over \, C_1 = C_0 + C_1(1 - \alpha_0)$$
$$\gamma = \alpha \, over \, \alpha = \alpha_0 + \alpha_1(1 - \alpha_0)$$

Where $C_{1,2}$ and $\alpha_{1,2}$ are the input opacity-weighted color and opacity values, respectively, and K and γ are the output opacity-weighted color and opacity values, respectively. Note that the OVER operator performs the same calculation on each of the red, green, and blue channels; we use a single value to simplify the discussion. We want two child impostors that, when composited together, produce the same affects are the parent impostor. Hence, K and γ correspond to the parent impostor, and $C_{1,2}$ and $\alpha_{1,2}$ correspond to the child impostors. We also simplify by making the front/back imagery the same, e.g., $C_1 = C_2$ and $\alpha_0 = \alpha_1$.

Solving for C, given K:

$$K = C + (1 - \alpha)C = C(2 - a)$$

thus

$$C = \frac{K}{(2 - \alpha)}$$

And solving for α, given γ:

$$\gamma = \alpha + \alpha(1 - \alpha) = 1 - (1 - \alpha)^2$$

thus

$$\alpha = 1 - \sqrt{1 - \gamma}$$

However, we have observed two issues with these formulas. First, these operations are performed on 8-bit integer values, and can be very error prone. Secondly, the structures in the data are not placed uniformly in the associated region. If, for example, the front half of the region associated with the parent impostor is empty, the child impostor associated with that front half will contain imagery belonging to the back half. This is not a problem until one of the children is re-rendered. If the front child is re-rendered, the overall contribution of the front/back regions will decrease as they effectively become more transparent. Similarly, if the back impostor is re-rendered first, the

# Processors	Rate			Speed-Up
	Ave	Min	Max	
1	14	11	20	1.0
2	27	21	39	1.9
4	63	52	79	4.5
8	114	104	125	8.1
16	239	217	257	14.9
32	407	360	451	29.0

Table 1: Scalability study on Origin3000 with 48 250Mhz R10000s. Mandelbrot tile size is 128^2 pixels. The rates listed are in tiles generated per second. This study does not consider rates for subdivide or merge requests, as they issued much less frequent that generate requests. The variation is rates is due to measuring an interactive session, rather than a scripted one, so there is some variation in what was generated.

overall contribution will increase, and they effectively become more opaque. Fortunately, due to proximity and viewing parameters, the front and back pairs are often re-rendered at nearly the same time.

Merge

Merge a left/right, top/bottom, or front/behind pair for form a lower resolution image. This is used when the camera position has not changed, but the region is less important (the user has turned the view frustum away). The left/right and top/bottom pairs are produced by low-pass filtering or subsampling, depending on which filtering mode they has originally used. Merging a front/back pair is simply compositing the front and back pairs together, e.g., $parent = front\,OVER\,back$.

Generate

Actually render a region to an image. VTK is used as the rendering engine, so anything that VTK can render can be rendered and cached in our system.

6 Results

We show two different sets of results. The first is a functionality study using the VTK-based visualization based on our shared-memory system; the second is scalability study using a multiresoultion Mandelbrot fractal viewer. Both use the same underlying infrastructure.

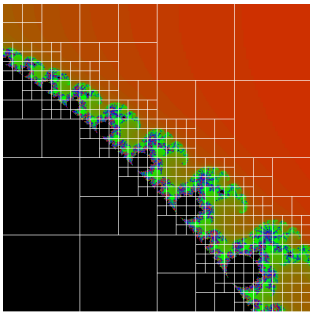


Figure 7: Example multiresolution Mandelbrot.

The reason that we have two is the nature of VTK. When using the hardware-texture volume visualization or geometry-based iso-surface rendering, running more than one generator process per an InfiniteReality3 pipe severely impacts performance - due entirely to the pipeline flushes that occur during context switches. Secondly, software versions of volume visualization and iso-surface rendering still require a display window on the desktop (this off-screen functionality does seem to work). This means that all generated imagery is written to the display, then read back, which is extremely slow. For the former, one cannot reasonably do a scalability study with only four pipes. For the latter, the system fails to scale beyond four processes. Our test of the VTK system allocated three pipes to generation, and one to viewing. Scaling was linear.

Our scalablility test software is a multiresolution Mandelbrot program, which uses the same infrastructure as the VTK system. The generator processes performs the Mandelbrot calculation entirely in software. Speed-up is approximately linear; see table 1. On a side note: the file that is used for shared memory can be displayed by our system without any generator processes, i.e., it can be used as a variety of multiresolution image file.

The timing results were collected as follows: every time a task is removed from the done queue by the viewer process, it increments a counter. Once a second, the counter is printed out, then reset to zero.

7 Future Work

Our multiresolution caching system shows promise, but there are a number of additions to before it become a useful product for scientists.

Currently, our system uses a round-robin approach to updating impostors. This is not sufficient when there are a large number of impostors being displayed. If there are a large number of impostors, the rate at which impostors are update is slow, and it can become difficult to navigate an environment. We plan to implement an error- and view-criterion-driven scheduler.

We plan to rewrite the rendering engine and scrap our use of VTK. The is considerable overhead rendering a frame in VTK, and it is not optimized for texture-based volume visualization. VTK's texture-based volume rendering is extremely inefficient and slow, especially when compared to what contemporary hardware-based volume visualization systems can do. Also, VTK's ray-casting engine has some issues with consistency of lighting calculations for the same scene when the camera is facing different directions.

Third, we plan to augment the system to handle multiresolution, time-vary datasets.

Last, we plan to move our system to cluster of Linux boxes. While the Origin3000's shared memory system performs very nicely, contemporary graphics cards are an order of magnitude faster than Infinite-Reality3 pipes.

References

[Bet00] Wes Bethel. Visualization viewpoints: Visualization dot com. *IEEE Computer Graphics and Applications*, 20(3):17–20, / 2000.

[BSL+00] W. Bethel, J. Shalf, S. Lau, D. Gunter, J. Lee, B. Tierney, V. Beckner, J. Brandt, D. Evensky, H. Chen, G. Pavel, J. Olsen, and B. H. Bodtker. Visapult — using high-speed WANs and network data caches to enable remote and distributed visualization. In ACM, editor, *SC2000: High Performance Networking and Computing. Dallas Convention Center, Dallas, TX, USA, November 4–10, 2000*, pages 118–119, New York, NY 10036, USA and 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 2000. ACM Press and IEEE Computer Society Press.

[CSKK99] Baoquan Chen, J. Edward Swan, II., Eddy Kuo, and Arie E. Kaufman. LOD-Sprite Technique for Accelerated Terrain Rendering. In David Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization '99 (Vis '99)*, pages 291–298, San Francisco, October 1999. IEEE, IEEE.

[CTFB01] Marcello Carrozzino, Franco Tecchia, Claudia Falcioni, and Massimo Bergamasco. Image caching algorithms and strategies for real time rendering of complex virtual environments. In Stephen N. Spencer, editor, *Prceedings of the 1st International Conference on Computer Graphics, Virtual Reality and Visualisation in Africa (AFRIGRAPH-01)*, pages 65–74, New York, November 5–7 2001. ACM Press.

[SLS+96] Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–82. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[SS96] G. Schaufler and W. Stürzlinger. A three-dimensional image cache for virtual reality. In *Proceedings of EUROGRAPHICS'96*, 1996.